About Docker :

Docker is an open-source software tool designed to automate and ease the process of creating, packaging, and deploying applications using an environment called a *container*. The use of Linux containers to deploy applications is called containerization. A Container allows us to package an application with all of the parts needed to run an application (code, system tools, logs, libraries, configuration settings and other dependencies) and sends it out as a single standalone package deployable via [Ubuntu](https://goo.gl/f77a2v) (in this case 16.04 LTS). Docker can be installed on [other](https://goo.gl/ZpfYKy) [platforms](https://goo.gl/iTES81) as well. Currently, the Docker software is maintained by the Docker community and Docker Inc. Check out the [official documentation](https://docs.docker.com/) to find more specifics on Docker. Docker Terms and Concepts.

Some terms related to Docker:

1. **Containerization:** Containerization is a lightweight alternative to full machine virtualization (like VMWare) that involves encapsulating an application within a container with its own operating environment.
2. **Docker Image:** A *Docker Image* is the basic unit for deploying a Docker container. A Docker image is essentially a static snapshot of a container, incorporating all of the objects needed to run a container.
3. **Docker Container:** A *Docker Container* encapsulates a Docker image and when live and running, is considered a container. Each container runs isolated in the host machine.
4. **Docker Registry:** The *Docker Registry* is a stateless, highly scalable server-side application that stores and distributes Docker images. This registry holds Docker images, along with their versions and, it can provide both public and private storage location. There is a public Docker registry called [Docker Hub](https://hub.docker.com/) which provides a free-to-use, hosted Registry, plus additional features like organization accounts, automated builds, and more. Users interact with a registry by using Docker push or pull commands.
5. **Docker Engine:** The *Docker Engine* is a layer which exists between containers and the Linux kernel and runs the containers. It is also known as the Docker daemon. Any Docker container can run on any server that has the Docker-daemon enabled, regardless of the underlying operating system.
6. **Docker Compose:** *Docker Compose* is a tool that defines, manages and controls multi-container Docker applications. With Compose, a single configuration file is used to set up all of your application’s services. Then, using a single command, you can create and start all the services from that file.

1. **Dockerfiles:** Dockerfiles are merely text documents (.yaml files) that contains all of the configuration information and commands needed to assemble a container image. With a Dockerfile, the Docker daemon can automatically build the container image.

Installing Docker-engine :

To Set Up Repository for Docker :

1. **sudo apt-get update**

To update the apt packages of the system to the latest version.

1. **sudo apt-get install \**

**apt-transport-https \**

**ca-certificates \**

**curl \**

**gnupg-agent \**

**software-properties-common**

Install packages to allow apt to use a repository over HTTPS.

For example, cloning a Github repository to the desktop over https.

3.  **curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -**

By using curl we are pulling a gpg file over https which contains the official docker gpg key. Curl command is used to pull a file over various protocols such as https, ftp, IP, etc.

4.  **sudo apt-key fingerprint 0EBFCD88**

Now we need to verify the key by searching the last 8 characters with the fingerprint.

5. **sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"**

To install Docker Engine :

1. **sudo apt-get update**

To update the apt packages of the system to the latest version.

1. **sudo apt-get install docker-ce docker-ce-cli containerd.io**

Installs the latest version of docker ce and container. As we have not provided any version explicitly it automatically installs the latest version.

We can install a specific version using command :

sudo apt-get install docker-ce=<VERSION\_STRING> docker-ce-cli=<VERSION\_STRING> containerd.io

3. Now to verify the successful installation of docker we can run a container .

**docker run hello-world**

If the image of the container is already present it creates an instance and runs the container otherwise pull the image of the container from the docker hub and runs the container.

Whenever we need to run a container we first need to pull its image from the docker hub to our local machine and then run the instance of the image(i.e container) on a port.

**Install Machine directly**

* Install Docker.
* Download the Docker Machine binary and extract it to your PATH.
* If you are running Linux:

**$base=https://github.com/docker/machine/releases/download/v0.16.0 &&curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine && sudo install /tmp/docker-machine /usr/local/bin/docker-machine**

* Check the installation by displaying the Machine version:

**$ docker-machine version**

**Install Compose on Linux systems**

* To download the current stable release of Docker Compose:

**sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose**

* To Apply executable permissions to the binary:

**sudo chmod +x /usr/local/bin/docker-compose**

* For example:

**sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose**

* To Test the installation.

**docker-compose --version**

# 

# **Open edX Ironwood Release**

1. The Open edX Ironwood release contains several new features for learners, course teams, and developers.
2. A git tag identifies the version of Open edX code that is the Ironwood release. About two dozen repositories are tagged as part of an Open edX release. Many other repositories are installed as dependencies of those repositories.
3. You can install the Open edX Ironwood release using either [Devstack](https://edx.readthedocs.io/projects/edx-installing-configuring-and-running/en/latest/installation/install_devstack.html#installing-the-open-edx-developer-stack) or the the [Open edX Native Installation](https://openedx.atlassian.net/wiki/x/g4G6C) instructions.

**Install Devstack:** Run these commands in terminal to install ironwood devstack .

* Fork the repository from <https://github.com/edx/devstack> to your own github account

Forking means creating a copy of the repository to our own account.

* **git clone** [**https://github.com/edx/devstack**](https://github.com/edx/devstack)

Clone the devstack github repository from your github account to your local system.

* **cd devstack**

The cd command changes our working directory to our newly cloned local repository.

* **git checkout open-release/ironwood.master**

The checkout command takes us to the ironwood branch of the repository.

We can move to any branch we wish to download. For example hawthorn.master, ginkgo.master or latest, which is the newest release of devstack.

* **export OPENEDX\_RELEASE=ironwood.master**

Set the environment variable OPENEDX\_RELEASE to ironwood.master to download the corresponding images for the services.

* **make dev.checkout**

To check out the correct branch in the local checkout of each service repository.

In the Makefile it calls the checkout function in repo.sh file. The checkout function makes an array containing all the positional arguments. For each cloned repository we move to the ironwood.master branch in each repo.

repos is a array of github repositories that are services that needs to be cloned. The images of these services are pulled when the dev.provision command is executed.

These services can be found in the docker-compose.yml file.

***repos=("https://github.com/edx/course-discovery.git"***

***"https://github.com/edx/credentials.git"***

***"https://github.com/edx/cs\_comments\_service.git"***

***"https://github.com/edx/ecommerce.git"***

***"https://github.com/edx/edx-e2e-tests.git"***

***"https://github.com/edx/edx-notes-api.git"***

***"https://github.com/edx/edx-platform.git"***

***"https://github.com/edx/xqueue.git"***

***"https://github.com/edx/edx-analytics-pipeline.git"***

***"https://github.com/edx/gradebook.git")***

Commands in file repo.sh

For creating the array :

***repos\_to\_checkout=("$@")***

For checking out to ironwood.master in each repo :

***for repo in "${repos\_to\_checkout[@]}"***

***do***

***[[ $repo =~ $name\_pattern ]]***

***name="${BASH\_REMATCH[1]}"***

***if [ -d "$name" -a -n "$(ls -A "$name" 2>/dev/null)" ]; then***

***echo "Checking out branch ${OPENEDX\_GIT\_BRANCH} of $name"***

***cd $name***

***\_checkout\_and\_update\_branch***

***cd ..***

***fi***

***done***

We have used the bash regex operator to capture the name of the repository and the match results are saved to the array named $BASH\_REMATCH.

Now if the directory exists and it is non-empty it is assumed to be cloned otherwise we checkout to ironwood.master of that repo and in next step it is cloned.

* **make dev.clone**

Clones the Open edX service repositories. The Docker Compose file mounts a host volume for each service’s executing code. The host directory defaults to be a sibling of the /devstack directory.

This command executes the dev.clone in Makefile which calls the clone function in repo.sh file. This function clones the all the repositories from the ironwood.master branch in the repos. The Open edX service repositories cloned are :

1. ecommerce
2. course\_discovery
3. credentials
4. cs\_comments\_service
5. edx-analytics-pipeline
6. edx-e2e-tests
7. edx-notes-api
8. edx-platform
9. gradebook
10. src
11. Xqueue

* **make dev.provision**

This script will provide all of the services.

The following commands in the provision.sh file is executed on running the dev.provision and each service will be set up :

* Bring the databases online.

**docker-compose up -d mysql mongo**

Compose pulls a MySQL and mongo image, builds an image for the code, and starts the services that are defined.

* Ensure the MySQL server is online and usable

**docker exec -i edx.devstack.mysql mysql -uroot -se "SELECT EXISTS(SELECT 1 FROM mysql.user WHERE user = 'root')"**

This command checks weather the root user exists or not.

* **docker exec -i edx.devstack.mysql mysql -uroot mysql < provision.sql**

This command will be used to give privileges for databases to following users:

(Database Name): (User Name)

Credentials: credentials001

Discovery: discov001

Ecommerce: ecomm001

Edxmktg: edxmktg001

Notes: notes001

Edxapp: edxapp001

Edxapp\_csmh: edxapp001

Then FLUSH PRIVILEGES will be used to reload grant table

For more info follow these links : [link1](https://stackoverflow.com/questions/31111847/identified-by-password-in-mysql) [link2](https://stackoverflow.com/questions/36463966/when-is-flush-privileges-in-mysql-really-needed)

* **docker exec -i edx.devstack.mongo mongo < mongo-provision.js**

conn = new Mongo();

users = [

{

'user': 'admin',

'pwd': 'password',

'roles': ['root'],

'database': 'admin'

},

{

'user': 'cs\_comments\_service',

'pwd': 'password',

'roles': ['readWrite'],

'database': 'cs\_comments\_service'

},

{

'user': 'edxapp',

'pwd': 'password',

'roles': ['readWrite'],

'database': 'edxapp'

}

];

for (var i = 0; i < users.length; i++) {

var user = users[i];

var username = user.user;

var db = conn.getDB(user.database);

delete user.database;

if (db.getUser(username) == null) {

db.createUser(user);

} else {

delete user.user;

db.updateUser(username, user);

}

}

It creates newer version of following users:

* Admin
* Cs\_comments\_service
* Edxapp

providing or supplying resources for the learning management system and central management system(studio).

./provision-lms.sh

On executing of the above command it executes the file provision-lms.sh.

This file first executes the load.sh file passed an argument i.e the name of the database to load.

***./load-db.sh edxapp***

***./load-db.sh edxapp\_csmh***

The load.sh on first loads the edxapp database and then the edxapp\_csmh database.

The load.sh file executes the command

***docker exec -i edx.devstack.mysql mysql -uroot $1 < $1.sql***

The above command docker exec is used to run on a running container.

-i flag opens it in an interactive mode (i.e keeps the stdin open if it is not attached).

If the root user is a root user it executes the edxapp.sql and creates the edxapp database.

Then we create container instances to bring the databases online.

***docker-compose $DOCKER\_COMPOSE\_FILES up -d $app***

Then the following steps are executed:

1. Installing prereqs crashes the process

***docker-compose restart lms***

1. Run edxapp migrations first since they are needed for the service users and OAuth clients

***docker-compose exec lms bash -c 'source /edx/app/edxapp/edxapp\_env && cd /edx/app/edxapp/edx-platform && paver update\_db --settings devstack\_docker'***

1. Create a superuser for edxapp

***docker-compose exec lms bash -c 'source /edx/app/edxapp/edxapp\_env && python /edx/app/edxapp/edx-platform/manage.py lms --settings=devstack\_docker manage\_user edx edx@example.com --superuser --staff'***

***docker-compose exec lms bash -c 'source /edx/app/edxapp/edxapp\_env && echo "from django.contrib.auth import get\_user\_model; User = get\_user\_model(); user = User.objects.get(username=\"edx\"); user.set\_password(\"edx\"); user.save()" | python /edx/app/edxapp/edx-platform/manage.py lms shell --settings=devstack\_docker'***

1. Create an enterprise service user for edxapp

***docker-compose exec lms bash -c 'source /edx/app/edxapp/edxapp\_env && python /edx/app/edxapp/edx-platform/manage.py lms --settings=devstack\_docker manage\_user enterprise\_worker enterprise\_worker@example.com'***

1. Enable the LMS-E-Commerce integration

**docker-compose exec lms bash -c 'source /edx/app/edxapp/edxapp\_env && python /edx/app/edxapp/edx-platform/manage.py lms --settings=devstack\_docker configure\_commerce'**

1. Create demo course and users

**docker-compose exec lms bash -c '/edx/app/edx\_ansible/venvs/edx\_ansible/bin/ansible-playbook /edx/app/edx\_ansible/edx\_ansible/playbooks/demo.yml -v -c local -i "127.0.0.1," --extra-vars="COMMON\_EDXAPP\_SETTINGS=devstack\_docker"'**

1. Fix missing vendor file by clearing the cache

***docker-compose exec lms bash -c 'rm /edx/app/edxapp/edx-platform/.prereqs\_cache/Node\_prereqs.sha1'***

1. Create static assets for both LMS and Studio

***for app in "${apps[@]}"; do***

***docker-compose exec $app bash -c 'source /edx/app/edxapp/edxapp\_env && cd /edx/app/edxapp/edx-platform && paver update\_assets --settings devstack\_docker'***

***done***

1. Provision a retirement service account user

***./provision-retirement-user.sh retirement retirement\_service\_worker***

For more detail follow the provision-lms.sh

Nothing special needed for studio

docker-compose $DOCKER\_COMPOSE\_FILES up -d studio

./provision-ecommerce.sh

./provision-discovery.sh

./provision-credentials.sh

./provision-e2e.sh

./provision-forum.sh

./provision-notes.sh

All the 15 docker images running after executing this make dev.provision

1. chrome
2. elasticsearch
3. firefox
4. memcached
5. mongo
6. mysql
7. credentials
8. discovery
9. ecommerce
10. lms
11. edx\_notes\_api
12. studio
13. forum
14. devpi
15. gradebook

* **make dev.up**

This command makes the containers for the pulled docker images and hosts them in different ports under the localhost.

The list of docker containers are : [Dockers List](https://drive.google.com/open?id=0B168SrnPp4BpZ1RCQ1VncU81VEZYanl2SUNWN2dtRUg1bGRv)

The log file for provision are : [provision.log](https://drive.google.com/open?id=0B168SrnPp4BpclQ3d3gwbDNWLWtrUzduYUt6Vko4S2h6bzBz)